Санкт-Петербургский государственный университет
Факультет прикладной математики - процессов управления
Кафедра компьютерного моделирования и многопроцессорных систем

Лапицкая Людмила Юрьевна

Выпускная квалификационная работа бакалавра

Разработка инструментария анализа сложносвязных социальных данных

Направление 010400
Прикладная математика и информатика

Научный руководитель,
ст. преподаватель
Кулабухова Н.В.

Санкт-Петербург
2016

Оглавление

Введение

За последние десятилетия огромное внимание стало уделяться связям, существующим в нашем обществе, что привело к появлению отдельного направления исследований. Ученые рассматривают социальные отношения в рамках теории сетей, в основе которой лежит математическая теория графов, а также эмпирические исследования в области различных общественных наук (антропологии, социологии, биологии и др.)[1]. Основные термины теории сетей включают в себя понятие **узла** (отвечает за отдельного участника в пределах сети) и **связи** (отображает различные отношения между участниками сети: дружеские, рабочие, учебные и другие) [1].

Данное направление исследований в последнее время переживает новый подъем благодаря появлению социальных онлайн-ресурсов таких как социальные сети, форумы и др. На сегодняшний день социальные сети (такие как Facebook, Вконтакте, Instagram, Одноклассники, Twitter и др.) являются одними из самых посещаемых сайтов во всем мире [2]. Доступность большого объёма персональной информации пользователей (интересы, мнения, связи с другими пользователями) открывает большие возможности для эффективного решения исследовательских и бизнес-задач, решение которых раннее было очень кропотливым, трудоёмким, а зачастую невозможным, что делает данное направление исследований актуальным. Исследованиями социальных данных активно занимаются университеты Стэнфорд [3, 4], Карнеги-Меллон, Оксфорд, INRIA, а также компании Facebook, Google, Yahoo!, LinkedIn и многие другие [5].

Под **анализом социальных данных** понимается их обработка, позволяющая определить организацию исходной сети, обобщая их по определенному критерию, и выявить какие-либо признаки, характеризующие

эти данные, такие как: интересы участников, их образ жизни, мнения по каким-либо вопросам и т.д.

Анализ социальных данных позволяет произвести моделирование распространения информации в сети, выявить характерные признаки поведения пользователя, помогает в развитии систем рекомендаций, а также прогнозировании связей. В частном секторе фирмы используют анализ социальных сетей для поддержки такой деятельности, как взаимодействие и анализ клиентов, маркетинг и бизнес-аналитика. Использование анализа социальных сетей государственным сектором включает в себя развитие стратегий участия руководства, использование средств массовой информации и основанное на сообществах решение проблем.

Основными сложностями в работе с социальными данными являются: их большой объем, многообразие, неоднозначное качество получаемой информации (недостоверная информация, спам, ложные аккаунты), которые делают процесс анализа вручную почти невозможным. К примеру, в социальной сети ВКонтакте на сегодняшний день зарегистрировано более 350 миллионов пользователей [6], а база данных социальной сети Facebook содержит более 1 миллиарда пользовательских аккаунтов и более 100 миллиардов связей между ними. Каждый день пользователи добавляют более 200 миллионов фотографий и оставляют более 2 миллиардов комментариев [5]. В связи с этим, возникает потребность в новых решениях, позволяющих осуществлять распределённую обработку данных, а также сделать её более доступной для большего количества исследователей.

Постановка задачи:

Целью данной работы является создание инструментария для сбора и обработки сложно связанных социальных данных, который по данному социальному графу и связанному с ним набору сообщений пользователей позволяет обнаружить сообщества среди участников сети, провести текстовую классификацию, разделив имеющиеся текстовые данные на категории, а также составить тематические модели для полученных категорий.

Процесс достижения поставленной цели можно представить в виде следующего **набора задач**:

1. Исследование предметной области
    - Изучение принципов работы и модели программирования GraphChi
    - Изучение существующих алгоритмов обнаружения сообществ, выбор оптимального решения и его обоснование
    - Выбор решения для классификации текстов и тематического моделирования на русском языке
2. Реализация выбранных методов:
    - Выбор средств для реализации
    - Программирование выбранных алгоритмов
3. Тестирование разработанного программного обеспечения
    - Сбор данных из социальной сети
    - Анализ полученных данных, подведение итогов

Обзор литературы

В связи с растущим интересом исследователей к задачам анализа социальных данных за последние годы было написано немало статей, освещающих разные аспекты данной проблемы. В работе [1] собраны основные термины и основополагающие понятия и утверждения теории социальных сетей.

В первой части данной работы рассмотрена задача обнаружения сообществ в социальных сетях. Рассматриваемые далее алгоритмы были имплементированы с помощью системы для распределенной обработки графов - GraphChi, основные принципы работы который описаны A. Kyrola в работе [7]. В статье [12] авторы сравнивают производительность работы алгоритмов обнаружения сообществ, реализованных на GraphChi и с помощью MapReduce. Первая программа была запущена на обычном персональном компьютере, вторая - на кластере Amazon EC2, и было установлено, что первая показала результаты, превосходящие вторую по производительности в 4-6 раз.

К.В. Воронцов в своей работе [8] описал основные существующие алгоритмы кластеризации. В работах [9, 11] рассмотрены алгоритмы обнаружения сообществ, основанные на удалении ребер с максимальной промежуточностью, которые были включены в разработку инструментария. В работе [13] У. Брандес предлагает алгоритм для эффективного вычисления промежуточности для вершин, который затем модифицирует для вычисления промежуточности многих других видов, в том числе, промежуточности ребер, что и было использовано в данной работе.

Далее была рассмотрена задача вычисления PageRank узлов в графе. В работах [15, 16, 17, 18, 19] рассмотрены различные подходы к решению данной задачи. В работах [15, 17] авторы предлагают алгебраические подходы для вычисления PageRank, которые дают точный результат

вычисления, но являются вычислительноёмкими. В работе [18] авторы рассматривают основные алгоритмы, основанные на методе Монте-Карло, которые позволяют аппроксимировать значение PageRank, при этом достигая значительных улучшений в быстродействии в сравнении с алгебраическими подходами. A.D. Sarma, A.R.Molla, G.Pandurangan, E. Upfal предложили алгоритм для распределенного вычисления значения PageRank, основанный на симуляции случайных блужданий, сложность которого составляет $O(\frac{log(n)}{\epsilon})$ [19], который был использован при создании данного инструментария.

В статьях [20, 21] описан алгоритм распространения меток для обнаружения сообществ. Авторы отмечают в качестве его преимуществ перед другими алгоритмами - простоту, а также низкую вычислительную сложность. Одной из проблем в реализации функции обнаружения сообществ в рамках данного инструментария было то, что модулярность (основная метрика качества для данной задачи) определена только для неориентированных графов, в то время как многие из социальных отношений являются ориентированными. В работе [22] предложено обобщение модулярности на ориентированные графы, которое было использовано в качестве решения данной проблемы.

Во второй части данной работы рассмотрено создание функционала для текстовой классификации и тематического моделирования сообщений пользователей социальной сети. Ключевые понятия и теория были получены из классической книги Daniel Jurafsky and James H. Martin по компьютерной обработке естественных языков [24]. Для реализации текстовой классификации на языке программирования Python были использованы работы [25, 27]. В первой из них описаны основные принципы работы с библиотеками для обработки естественных языков в Python. Во второй

описан принцип работы мультиномиального байесовского классификатора, а так же различных его модификаций. Также источником информации при создании данной функции инструментария послужили работы К. Воронцова [29, 32], в которых подробно описаны наиболее распространенные на данный момент тематические модели, принципы их работы, их достоинства и недостатки.

Глава 1. Обнаружение сообществ

1.1. Обзор предметной области

Обнаружение сообществ (или *кластеризация*) в графах является одной из наиболее изучаемых задач в области анализа социальных данных, так как имеет большое количество приложений в реальном мире. Сообщества хорошо отображают отношения между сущностями, позволяя обнаруживать определенные паттерны, возникающие в социальных сетях, а также позволяют гораздо более эффективно визуализировать социальные данные. В качестве других целей кластеризации можно выделить:

1. Понимание структуры социальной сети путём разбиения её на подгруппы;
2. Упрощение дальнейшей работы с данными, рассматривая каждое сообщество по отдельности (стратегия «разделяй и властвуй»);
3. Сокращение объема хранимых данных в случае сверхбольшой выборки, оставляя наиболее типичных представителей каждого класса.

Однако обнаружение сообществ в крупномасштабных сетях является нетривиальной задачей, так как её решение предполагает наличие больших вычислительных мощностей. Тем не менее, за последние годы было разработано достаточно много алгоритмов кластеризации социальных графов. Наиболее распространённые среди них можно объединить в три большие группы:

1. Иерархические алгоритмы кластеризации
2. Спектральные алгоритмы кластеризации
3. Алгоритмы максимизации модулярности

Иерархические алгоритмы кластеризации, называемые также алгоритмами таксономии, строят систему вложенных разбиений выборки на классы (в отличие от других алгоритмов, строящих одно разбиение). Среди алгоритмов иерархической кластеризации различаются два основных типа. Дивизимные (или *нисходящие*) алгоритмы разбивают выборку на всё более и более мелкие кластеры. Более распространены агломеративные (или *восходящие*) алгоритмы, в которых объекты объединяются во всё более крупные кластеры.[8]

Также к этой группе относятся алгоритмы, использующие промежуточность ребра. Вместо обнаружения рёбер, находящихся внутри сообществ в графе, они пытаются найти рёбра *между* сообществами, и постепенно удаляют их один за одним, выделяя структуру сети. [11]

Спектральной кластеризацией называют все методы, которые разбивают множество на кластеры с помощью собственных векторов матрицы сходства S или других матриц, полученных из нее. Спектральная кластеризация состоит из преобразования исходного множества объектов в множество точек в пространстве, координатами которых являются элементы собственных векторов. Полученное множество точек затем кластеризуется с помощью стандартных методов, например *k*-средних.

Один из недостатков данных алгоритмов состоит в том, что для их работы необходимо знать количество сообществ заранее (иерархическая кластеризация данным недостатком не обладает).

Алгоритмы, основанные на максимизации модулярности. **Модулярность** - это мера, представляющая собой разность между долей рёбер внутри кластера в рассматриваемой сети и ожидаемой долей рёбер внутри кластера в сети, в которой вершины имеют ту же степень, что и в исходной, но рёбра распределены случайно (т.н. *конфигурационной модели*). Математически это выражается следующей формулой:

$$Q = \frac{1}{(2m)} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{(2m)} \right] \delta(c_v, c_w) = \sum_{i=1}^{c} (e_{ii} - a_i^2) \quad (1.1)$$

где

A - матрица смежности,

m - количество рёбер в графе,

k_v, k_w - степени вершин v и w соответственно,

δ - дельта Кронекера (показывает, находятся ли вершины v и w в одном сообществе) [9]

Однако конфигурационная модель предполагает, что каждая вершина может быть соединена с любой другой вершиной в сети с одинаковой вероятностью. Но для больших сетей это предположение не в полной мере справедливо, так как каждая вершина в этом случае соединяется с небольшой группой других вершин, большее же количество узлов в сети ими игнорируется. Это приводит к тому, что при увеличении размера сети ожидаемое количество рёбер между двумя кластерами в конфигурационной модели уменьшается и может даже быть меньше единицы. Если такое происходит, единственное ребро между двумя сообществами будет интерпретировано алгоритмами как знак сильной связи между ними, и максимизация модулярности приведет к слиянию этих кластеров. Поэтому одним из самых больших недостатков в применении алгоритмов максимизации модулярности является невозможность обнаружения достаточно малых сообществ внутри большой сети (*resolution limit*).

1.2. Система для распределённых вычислений на графах GraphChi

Существующие на данный момент системы для вычисления на графах требуют больших вычислительных кластеров или систем распределенных вычислений для обработки задач из реального мира (таких как анализ социальных данных). Ввиду того, что распределенные ресурсы все еще остаются труднодоступными большинству исследователей, а также учитывая то, что разработка распределенного программного обеспечения - непростая задача, особенно для людей далеких от области IT, для решения поставленной задачи было решено использовать GraphChi - систему для параллельных вычислений на графах, использующую ресурсы жесткого диска компьютера.[7]

Предполагается, что динамическая память с произвольным доступом (ДППД) имеющегося в распоряжении компьютера ограничена, и:

1. Структура графа, значения ребер и вершин не помещаются целиком в память
2. Вместительности ДППД достаточно, чтобы загружать ребра и значения каждой вершины в память по отдельности

Реализованный в системе подход заключается в разбиении графа на **подграфы** и **исполняемые интервалы (**непересекающиеся подмножества вершин**)** для уменьшения размерности задачи (что является стандартным приёмом в обработке больших графов), а также в использовании нового метода “параллельных скользящих окон” (PSW). [7] С его помощью можно обрабатывать графы с изменяемыми значениями как вершин, так и ребер, используя жесткий диск и малое количество не последовательных к нему обращений, в тоже время обеспечивая асинхронную модель вычислений. Подграфы обрабатываются PSW методом в три шага:

1. Загрузка подграфа с диска
2. Обновление значений вершин и ребер
3. Запись обновленных результатов на диск

После того как подграф для вершин интервала *p* полностью загружен в память, PSW параллельно выполняет определённую пользователем функцию обновления для вершины (update-function). Так как функция обновления может изменять и значения рёбер, для того, чтобы предотвратить конкурентное изменение одного и того же ребра смежными вершинами (состояние гонки), в GraphChi определён внешний **детерминизм исполнения**, который гарантирует, что каждый запуск PSW будет давать один и тот же результат. Это достигается следующим образом: вершины, которые способны порождать состояние гонки, помечаются как *критические* и обрабатываются последовательно. Не критические вершины (не соединены ребром с другими узлами в исполняемом интервале) могут быть безопасно обработаны параллельно. Необходимо отметить, что при обновлении критической вершины программе будут видны предыдущие изменения значений рёбер, что согласуется с асинхронной моделью программирования. Данное решение, конечно, уступает по эффективности полноценной параллельной обработке, поэтому для алгоритмов, не требующих непротиворечивости результатов, в GraphChi есть возможность отключить детерминизм.

Как было описано выше, программы, написанные для GraphChi реализуют **вершинно-ориентированную модель** программирования (Эта модель была воплощена на нескольких платформах для параллельных вычислений на графах, включая Pregel и GraphLab), поэтому наша задача сводится к реализации выбранных алгоритмов в вершинно-ориентированном стиле (в виде функции обновления для каждой вершины).

Итак, предлагаемая реализация обнаружения сообществ на GraphChi, будет состоять из 5 этапов, каждый из которых будет описан далее:

1. Обнаружение и удаление рёбер с наибольшей промежуточностью с помощью алгоритма Гирвана-Ньюмана
2. Вычисление PageRank для каждой вершины (для подсчёта LinkRank и обнаружения наиболее влиятельных членов сообществ)
3. Применение алгоритма распространения меток (Label Propagation Algorithm)
4. Вычисление обобщённой модулярности, основанной на LinkRank, в качестве метрики качества
5. Вывод в формате txt и gdf (для удобства визуализации)

1.3. Алгоритм Гирвана-Ньюмана и его реализация на GraphChi

Для реализации обнаружения сообществ на GraphChi был выбран алгоритм Гирвана-Ньюмана [11], так как в отличие от многих других алгоритмов он не требует хранения целого графа в ДППД и достаточно просто перекладывается на вершинно-ориентированную модель программирования. Данный метод принадлежит к группе алгоритмов разделяющей иерархической кластеризации. Алгоритм Гирвана-Ньюмана использует понятие промежуточности ребра, которое было упомянуто в п. 1.1.

Промежуточность ребра есть количество проходящих через него кратчайших путей между любой парой вершин в графе. Учитывая то, что сообщества в социальных сетях очень слабо соединены между собой

небольшим количеством меж-групповых ребер, все кратчайшие пути между сообществами должны проходить через одно из них. Тогда эти ребра, соединяющие сообщества, будут иметь высокое значение промежуточности. Таким образом, сообщества выявляются путем последовательного удаления таких ребер.

Алгоритм Гирвана-Ньюмана состоит из 4 этапов:

1. Вычисление промежуточности всех рёбер в сети.
2. Удаление рёбер с наивысшим значением промежуточности.
3. Пересчитывание значений промежуточности для новой сети.
4. Повторение шага 2 до достижения желаемого результата.

Основным **недостатком алгоритма**, несмотря на его частое использование, является его существенное ограничение в поддержке широкомасштабных сетей, так как требуется вычислять кратчайшие пути между каждой парой вершин в сети. Однако его параллельная реализация на GraphChi поможет сделать алгоритм гораздо более эффективным.

Функция обновления будет разделена нами на 4 этапа:

1. Инициализация - присвоение вершинам и рёбрам начальных значений для текущей итерации.
2. Вычисление кратчайших путей между вершинами.
3. Вычисление значений промежуточности рёбер на основе результатов стадии 2.
4. Удаление рёбер с наибольшим значением промежуточности.

1.3.1. Нахождение кратчайших путей и вычисление промежуточности

Для вычисления промежуточности воспользуемся алгоритмом, предложенным У. Брандесом [13], стабильно показывающим хорошие результаты в обработке графов большой размерности.

Рассмотрим ориентированный граф $G = (V, E)$, состоящий из множества вершин V и множества $E = V \times V$ ориентированных рёбер. Для ориентированного ребра $e = (u, v) \in E$, v будем называть начальной вершиной e, u - конечной вершиной, и будем считать, что v и u - смежные вершины.

Путь от вершины-источника $s \in V$ до вершины $t \in V$ - такая последовательность вершин и рёбер $s, (s, v_1), v_1, (v_1, v_2), v_2, \ldots, (v_k, t), t$, начинающаяся с s и заканчивающаяся t, что для каждой вершины имеется ребро, соединяющее её со следующей вершиной в последовательности. Длина (s, t)-пути - число рёбер, которое он содержит, а расстояние $dist(s, t)$ от s до t определяется как минимальный по длине путь (s, t).

Обозначим $\sigma(s, t)$ - число кратчайших путей (s, t), и $\sigma(s, t|e)$ - число кратчайших (s, t)-путей, проходящих через ребро e. Тогда **промежуточностью** ребра e будем называть величину

$$C_B(e) = \sum_{s, t \in V} \frac{\sigma(s,t|e)}{\sigma(s, t)}$$

где $\frac{0}{0} = 0$ по договорённости.

Эффективность вычислений достигается за счет следующих доказанных У. Брандесом соотношений:

Обозначим $\delta(s, t|e) = \frac{\sigma(s, t|e)}{\sigma(s, t)}$, $\delta(s|e) = \sum_{t \in V} \delta(s, t|e)$, тогда

$$\delta(s|e) = \sum_{(v, w) \in E,\, w:\, dist(s, w) = dist(s,v) + 1} \frac{\sigma(s, v)}{\sigma(s, w)}(1 + \delta(s, w)) \qquad (1.3.1)$$

Отсюда получаем, что $C_B(e) = \sum_{s \in V} \delta(s|e)$.

Алгоритм вычисляет промежуточность проходом по всем вершинам s $\in$ V , каждый раз вычисляя δ(s|e) для всех $v \in V$ в два шага. Первый шаг - поиск в ширину. На втором шаге проход осуществляется по всем вершинам в обратном порядке, т.е. вершины наиболее далекие от s, обрабатываются первыми, и величины, описанные в уравнении (1.3.1), складываются. Основной объём вычислений в алгоритме Брандеса приходится на поиск в ширину.

Вычислительная сложность алгоритма Брандеса $O(n(S(n, m) + m))$, где *S(n, m)* – сложность поиска кратчайших путей от одной вершины. Такая оценка получается исходя из того, что для каждой из *n* вершин требуется найти кратчайшие пути (со сложностью *S(n,m)*) и затем вычислить зависимости и обновить центральность (со сложностью *O(m)*).

Для алгоритма поиска в ширину *S(n,m) = O(m)*, и общая сложность составляет *O(nm)*. [14]

Для того, чтобы найти кратчайшие пути между каждой парой вершин в сети, на каждой итерации *i* будем обозначать вершину v_i за *источник* и находить все кратчайшие пути от неё до остальных узлов в графе.

Так как в GraphChi разрешено использовать пользовательский тип данных для вершины и ребра, мы создадим для них две структуры: VDataType и EDataType, которые будут описаны ниже. Вершина хранит два следующих элемента:

• *dist* показывает *длину* кратчайшего пути от текущей вершины-источника к себе.

• *sigma* показывает *количество* кратчайших путей от источника к себе.

В EdataType хранятся 6 следующих элементов:

• *edge_dist* показывает значение *dist,* передаваемое вершиной своим смежным вершинам.

• *edge_sigma* показывает значение *sigma,* передаваемое вершиной своим смежным вершинам.

• *visited* показывает, посещали ли данное ребро, и по умолчанию = *False.*

• *iteration_eb* показывает значение промежуточности ребра относительно текущей вершины-источника. Значение обнуляется, когда вершина-источник изменяется.

• *final_eb* показывает накопленное значение промежуточности ребра. Изначально оно = 0 и затем увеличивается на величину *iteration_eb,* когда вершина-источник изменяется.

• *deleted* показывает, удалено ли ребро (необходимо для работы программы в режиме *in-memory* для маленьких графов, так как в GraphChi рёбра могут быть удалены полностью только при перезаписи подграфа на диск, если же мы весь граф держим в памяти, то перезаписи не происходит, поэтому нам нужна метка, показывающая удалено ли ребро)

• *was_processed* показывает, было ли ребро обработано на стадии подсчёта промежуточности.

На шаге **инициализации** вершинам и рёбрам присваиваются начальные значения следующим правилам:

- Если вершина - источник, её *dist* = 0, *sigma* = 1, все соседние вершины планируются для обработки на следующей стадии. Если у текушей вершины-источника нет исходящих рёбер, то происходит реинициализация с новой вершиной-источником
- Если вершина не источник, её *dist* = ∞, *sigma* = 0
- Для любого ребра значения *edge_dist* и *edge_sigma* устанавливаются равными соответствующим значениям *dist* и *sigma* их начальных вершин, значения *deleted*, *visited*, *was_processed* = *False*, значения *iteration_eb* и *final_eb* = 0.

На шаге **подсчёта кратчайших путей** от вершины-источника *s* первыми будут обрабатываться параллельно вершины, смежные с источником, запланированные нами на шаге инициализации.

Для каждой вершины рассмотрим все входящие в нее ребра:

- Если ребро $e = (u, v)$ не было удалено ранее и не было посещено, то устанавливаем *visited* = *true* и сравниваем поля *dist* вершин *u* и *v* (к *v* имеем доступ напрямую, к *u* - через *edge_dist + 1* ребра *e*)
- Если *v.dist* > *u.dist (= e.edge_dist + 1)*, тогда *v.dist* = *u.dist*, *v.sigma* = *u.sigma* (т.е. *e.edge_sigma*), а также копируем новые значения *dist* и *sigma* вершины *v* всем исходящим из неё рёбрам $e = (v, t_i)$, а также если данное ребро еще не было посещено и не является удаленным, а вершина t_i не источник, то планируем её для последующей обработки
- Если *v.dist* = *u.dist,* мы нашли другой путь длины равной текущему кратчайшему пути, инкрементируем только значение *v.sigma* = *v.sigma* + *u.sigma* и копируем новое значение всем исходящим рёбрам

- Определяем концевые вершины графа, так как они первыми будут обработаны на следующей стадии

На шаге **вычисления промежуточности ребер** ее значение находится с использованием кратчайших путей, обнаруженных на прошлой стадии. Концевые вершины (листья) первыми планируются для обработки. Для вершины *v* и каждого входящего ребра *e = (u, v)* в кратчайшем пути, значение *iteration_eb* ребра *e* вычисляется по формуле Брандеса (1.3.1), переложенной на GraphChi (1.3.2).

$$iteration_{eb} = \frac{edge_{sigma}}{v.sigma}(1 + \sum_{e=(v,\, t_i)} iteration_{eb}) \qquad (1.3.2)$$

Затем текущее значение *iteration_eb* прибавляется к значению *final_eb*, чтобы в итоге получить конечное значение, при котором все вершины побывали источниками.

После этого вершины, смежные с текущей вершиной через ее входящие ребра, входящие в кратчайшие пути до источника, планируются для дальнейших операций.

После того как обе стадии для конкретной вершины-источника завершились, следующая вершина становится источником, и над ней проводятся те же самые операции.

Выбор и удаление рёбер.

Гирван и Ньюман [11] показали, что между двумя сообществами на одной итерации гарантированно хотя бы одно ребро будет иметь высокую промежуточность (но не факт, что *все* межгрупповые ребра будут также иметь высокое значение промежуточности). Именно поэтому авторы акцентировали внимание на том, что для правильной работы алгоритма

необходимо удалять по *одному* ребру за одну итерацию, после чего промежуточность должна быть пересчитана заново. Однако Гирван и Ньюман рассматривали достаточно маленькие сети, и соответственно количество сообществ (а также межгрупповых ребер) было невелико. Мы же будем рассматривать большие сети, в которых огромное количество межгрупповых рёбер, многие из которых будут иметь высокое значение промежуточности, а значит удаление всего одного ребра за итерацию во много раз замедлит скорость работы. Это позволяет предположить, что удаление нескольких рёбер за итерацию в больших сетях незначительно скажется на качестве работы программы. В статье [12] авторы показали, что при удалении фиксированного количества ребер (>1) на каждой итерации время работы программы уменьшается в 4 раза, при этом теряется 10% точности, и уменьшается в 10 раз ценой 20% точности.

В данной работе было решено удалять не фиксированное количество ребер за итерацию (так как количество рёбер постоянно уменьшается и удалять каждый раз одинаковое количество нецелесообразно), а рёбра, промежуточность которых находится на 99,99 перцентиле (т.е. 0,01 процент ребер с максимальной промежуточностью). Было установлено, что это помогает уменьшить возникающую ошибку и значительно ускорить работу программы. Рёбра на данном этапе удаляются либо средствами GraphChi (если граф не помещается в ДППД, и программа запущена в режиме *disk-based*) при перезаписи подграфа, содержащего удаляемое ребро, на диск (в данном случае удаленное ребро просто не записывается), либо, если программа работает в режиме *in-memory*, то поле *deleted* ребра помечается как *true,* что учитывается на последующих этапах работы алгоритма.

1.4. Вычисление PageRank

1.4.1. Определение и краткий обзор существующих подходов

PageRank [15] - один из алгоритмов, используемый компанией Google для ссылочного ранжирования веб-страниц всемирной паутины (или иначе ранжирования вершин ориентированного графа). PageRank страницы представляет собой число, показывающее её релевантность на основе предположения о том, что страница является важной, если на неё указывают другие важные страницы.

Однако, несмотря на то, что изначально понятие было введено для веб-сайтов, оно является достаточно широким и применимо для любого графа или сети. Например, PageRank часто используется в задачах химии, биоинформатики (ProteinRank, GeneRank и др.), нейробиологии, литературы (BookRank), в рекомендательных системах (ItemRank), а также в анализе сложно связанных социальных данных [16]. В этой работе PageRank будет использоваться для вычисления LinkRank (аналог PageRank для ссылок) и выявления наиболее влиятельных членов сообществ.

Математически, PageRank есть вероятность того, что определённая страница будет посещена гипотетическим пользователем, который случайно переходит по гиперссылкам на веб-страницах, описываемая следующей формулой:

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n)), \qquad (1.4.1)$$

где

A - страница, для которой вычисляется PageRank

T1...Tn - страницы, ссылающиеся на *A*

d - параметр затухания от 0 до 1, который обычно устанавливают равным 0.85

$C(T_i)$ - количество ссылок на другие страницы на ресурсе T_i.

PageRank формирует распределение вероятностей на пространстве веб-страниц, соответственно сумма значений PageRank для всех узлов равна единице. Аналитически вычислить закон распределения PageRank довольно сложно, поэтому было создано множество алгоритмов для приближенной оценки этого распределения.

Существует несколько различных алгоритмов для вычисления PageRank, которые можно объединить в две большие группы:

1. Алгоритмы, использующие методы линейной алгебры
2. Алгоритмы, использующие методы Монте-Карло

Суть **алгебраических подходов**, в основном, сводится к решению следующего уравнения относительно π:

$$\pi^T = \pi^T G \qquad (1.4.2)$$

иными словами, требуется найти собственный вектор, соответствующий числу $\lambda = 1$ стохастической Google-матрицы *G*, элементы которой определяются следующим образом:

$$G_{ij} = \alpha \frac{w_{ij}}{w_i^{out}} + \frac{1}{N}(\alpha a_i + 1 - \alpha) \qquad (1.4.3)$$

где

α - 1-*d*, где *d* - параметр затухания, описанный в *(1.4.1)*,

w_{ij} - элементы матрицы смежности для исходного графа,

w_{out}^i - количество исходящих рёбер для вершины *i,*

$a_i = 1$ тогда и только тогда, когда вершина *i* не имеет исходящих рёбер, иначе $a_i = 0$.

Наиболее часто используемые алгебраические методы:

1. Power Iteration - широко используемый во многих задачах метод нахождения наибольшего по модуля собственного числа и соответствующего ему собственного вектора матрицы. Не использует

разложение матрицы, благодаря чему успешно применяется даже для очень больших разреженных матриц, однако может сходиться очень медленно. С. Брин и Л. Пейдж предложили именно его для нахождения PageRank в своей статье [15]

2. Решение СЛАУ:
    - Метод Якоби - один из методов простой итерации для решения СЛАУ
    - Метод Гаусса-Зейделя - итерационный метод для решения СЛАУ

В статье [17] было показано, что метод Гаусса-Зейделя сходится гораздо быстрее Power Iteration/метода Якоби, однако требует предварительной сортировки рядов матрицы A.

Идеей, лежащей в основе **алгоритмов Монте-Карло**, является приближение значения PageRank путём симулирования случайных блужданий и последующей оценки распределения на основе полученных данных. В работе [18] рассмотрены основные алгоритмы, использующие метод Монте-Карло для вычисления PageRank, а также экспериментально установлено, что данные методы позволяют довольно хорошо приблизить значение PageRank достаточно релевантных страниц уже после первой итерации. Здесь наблюдается основное различие данного подхода с основанным на методах линейной алгебры (в частности Power Iteration), которые после первой итерации принимают во внимание только лишь взвешенную сумму весов входящих рёбер. В числе других преимуществ данных методов указывается их предрасположенность к параллельной реализации и хорошая масштабируемость. Именно поэтому для данного инструментария был выбран один из алгоритмов этой группы.

1.4.2. Описание выбранного алгоритма и его реализация на GraphChi

Алгоритм, выбранный для реализации на GraphChi, описан в работе [19]. Авторы предоставили два распределённых метода вычисления PageRank на основе симуляции случайных блужданий. Первый из них может быть применим как для ориентированных, так и не ориентированных графов, и его **сложность** составляет $O(\frac{logn}{\varepsilon})$, где n - количество вершин в сети, 1 - ε - параметр затухания в (1.4.1). Второй - улучшенный алгоритм для вычисления PageRank, его сложность - $O(\frac{\sqrt{logn}}{\varepsilon})$, однако он применим только для неориентированных графов, что не подходит для решения поставленной задачи. Также алгоритм не требует прямой коммуникации между не соседними вершинами, что позволяет легко переложить его на модель программирования в GraphChi.

Основная идея предложенного метода описывается следующим образом: необходимо осуществить $K = clog(n), (c = \frac{2}{\delta'\varepsilon}$ δ' будет описано ниже) параллельных случайных блужданий от каждой вершины в сети. На каждой итерации любое такое блуждание завершается с вероятностью 1 - ε в случайной соседней вершине, и с вероятностью ε - в текущей вершине. Так как ε - вероятность завершения блуждания для каждой итерации, ожидаемая длина каждого блуждания $1/\varepsilon$. Пусть для каждой вершины v ведётся подсчёт числа её посещений через случайные блуждания (обозначим его как ζ_v). После завершения всех блужданий в сети, для каждой вершины делается оценка её PageRank в виде: $\pi = \frac{\zeta_v \varepsilon}{nK}$. (1.4.4)

Константа δ' , упомянутая раннее, может быть найдена из выражения:

$\delta' = 1 + t(1+\delta) - E[e^{tW}]$, где $W = \varepsilon Y$, Y - случайная величина, подчиняющаяся геометрическому распределению с параметром ε, оптимизацией его по t.

Изложенная идея подсчёта количества посещений - стандартная для алгоритмов аппроксимации PageRank, однако в данном алгоритме в отличие от многих других не используется прямого сообщения между не смежными вершинами. Также в рассмотренной работе было показано, что в сети не будет затора, даже если будет начато полиномиальное количество параллельных случайных блужданий от каждой вершины.

Описание реализации алгоритма на GraphChi:

1. Каждая вершина хранит 2 значения:
   - *couponCount,* отвечающее за количество “купонов” для случайного блуждания, которое имеет эта вершина
   - *zetaCount,* отвечающее за число её посещений из других вершин через случайные блуждания

Обе эти величины будут храниться в полях *dist* и *sigma* определенного раннее типа VDataType.

2. Проводится $\frac{Blogn}{\varepsilon}$ итераций, где B - достаточно большая константа, и на каждой из них выполняется следующее (параллельно для $\forall v \in V$, для которой *couponCount* не равен нулю):

2.1. У каждой вершины u смежной с v устанавливаем $T_v^u = 0$, это значение мы будем хранить в поле *edge_dist EdataType* исходящих из v ребер

2.2. $couponCount_v$ раз с вероятностью $1 - \varepsilon$ выбираем случайное исходящее из v ребро (v, u) , и для него инкрементируем значение T_v^u (*edge_dist)*, посещенные вершины планируем для обновления.

2.3. Для каждой вершины u пересчитывается:

- $zetaCount = zetaCount + \sum_{v \in N(u)} T_v^u$
- $couponCount_u = \sum_{v \in N(u)} T_v^u$

3. Подсчитываем PageRank каждой вершины по формуле:

$$\pi = \frac{zetaCount_v \varepsilon}{nK} \quad (1.4.4)$$

1.5. Алгоритм распространения меток (Label Propagation Algorithm)

В данной работе LPA будет использоваться совместно с алгоритмом Гирвана-Ньюмана для обнаружения сообществ. Изначально этот метод был предложен для простановки недостающих лейблов в задачах частичного обучения (semi-supervised learning). [20] Суть этого подхода состоит в том, что не помеченные вершины итеративно перенимают метки большинства своих соседей. В задачах обработки сложно связанных социальных данных метод был успешно применен для обнаружения сообществ [21]. Было решено скомбинировать этот алгоритм с алгоритмом Гирвана-Ньюмана для уменьшения количества итераций первого и увеличения точности второго. Было установлено, что на графах с около 3000 вершин достаточно хорошие результаты наблюдаются уже за 2-3 итерации алгоритма.

Суть алгоритма:

Каждой вершине присваивается уникальная метка, и на каждой итерации алгоритма эти метки изменяются на те, которые имеют большее количество вершин-соседей. Если же эти количества равны, то метка из них выбирается случайно. Этот метод достаточно близко моделирует образование сообществ в реальной жизни, поэтому тесно связанные группы вершин быстро принимают одну общую метку, особенно если межгрупповые ребра заранее удалены с помощью алгоритма Гирвана-Ньюмана. В конце алгоритма вершины, имеющие одинаковые лейблы, группируют вместе как сообщества. В качестве **преимуществ** данного алгоритма можно выделить его простоту, низкую вычислительную сложность, а также то, что знать количество

сообществ и их размер заранее не нужно. Однако в отличие от всех остальных алгоритмов обнаружения сообществ результаты, полученные с помощью LPA, могут отличаться при каждом запуске. Для того, чтобы сузить возможное количество решений, мы будем использовать видоизмененный алгоритм LPA, который будет описан ниже.

1.5.1. Описание метода и его реализация на GraphChi

Алгоритм:

1. На первом шаге каждому узлу присваивается начальная метка, равная его ID в GraphChi
2. За счёт асинхронности модели вычисления в GraphChi, мы можем параллельно обрабатывать вершины (результат обновления сразу же доступен другим вершинам), и для каждой из них определять новый лейбл по следующему правилу:

   2.1. Пусть для $\forall\ v \in V$, $N(v)$ − множество смежных с ней вершин, $C = \{C_1^v, C_2^v, ..., C_n^v\}$ - множество меток вершин из $N(v)$. Тогда множество соседей вершины v, имеющих метку C_i^v обозначим за $N_i(v)$ $:\cup N_i(v) = N(v)$, а также пусть $B_i(v)$ - подмножество $N_i(v)$ такое, что $u \in B_i(v)$, тогда и только тогда, когда

   $\exists$ ребра $e_1 = (v, u) \in E$ и $e_2 = (u, v) \in E$.

   Тогда на следующем шаге алгоритма для каждой из C_i^v вычисляем

$$score(C_i^v) = |N_i(v)|\ (1 + \frac{\sum_{u \in N_i(v)} PageRank(u)}{|N_i(v)|})(1 + \frac{|B_i(v)|}{|N_i(v)|})$$

   2.2. В качестве нового лейбла для вершины выбираем

$$C^v = argmax(score(x)),\ x \in C$$

3. Алгоритм работает до тех пор, пока значения меток изменяются

Предложенный подход позволяет распространить лейблы за меньшее количество итераций, и, благодаря использованию формулы выше вместо случайного выбора метки в случае равных значений $|N_i(v)|$, предложенного в работе [21], помогает сильно уменьшить количество возможных результатов работы программы, делая её более детерминированной.

1.6. Обобщение модулярности на ориентированные графы и LinkRank

Стандартной метрикой качества в задаче обнаружения сообществ является модулярность, предложенная Ньюманом [9]. Однако данная функция определена только для неориентированных графов. Но очень многие социальные сети представляют из себя именно ориентированные графы (например, Twitter, в которых ребро - отношение follow, которое может быть не взаимно). В таких сетях направление ребра содержит в себе важную информацию, такую как асимметричное влияние или направление потока информации. Именно поэтому подходы, в которых направление рёбер игнорируется, очень не точно описывают реальную структуру сети.

В качестве метрики качества работы алгоритма нами будет использоваться обобщение модулярности, основанное на LinkRank [22]. В рассматриваемой работе авторы предлагают обобщение модулярности, основанное на LinkRank - аналоге PageRank для рёбер (показывает важность ребра в сети). Аналогично определению PageRank, LinkRank определённого ребра есть вероятность того, что гипотетический пользователь пройдёт по ссылке, ведущей от узла *i* к узлу *j*. Тогда, пользуясь определениями π и G

(1.4.1, 1.4.2), LinkRank может быть определён как: $L_{ij} = \pi_i G_{ij}$, где π_i - i-ый элемент PageRank-вектора, G_{ij} - элемент Google-матрицы G. Как уже было упомянуто в параграфе 1.1, модулярность в не ориентированных графах определяется следующим образом:

$$Q = \frac{1}{(2m)} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{(2m)} \right] \delta(c_v, c_w) = \sum_{i=1}^{c} (e_{ii} - a_i^2)$$

Иначе, модулярность есть разность между долей рёбер внутри сообществ и ожидаемой величиной этой доли. Авторы же предлагают новое определение модулярности для ориентированных графов:

Q^{lr} = доля времени, проведенного случайным блуждателем внутри сообществ - ожидаемое значение этой доли, или иначе:

$$Q^{lr} = \sum_{i,j} (L_{ij} - \pi_i \pi_j) \delta_{c_i c_j}$$

Было показано, что для не ориентированных графов новое определение модулярности полностью совпадает со старым.

1.7. Тестирование работы алгоритмов

1.7.1. Результаты обнаружения сообществ

Для имплементации всех алгоритмов была использована C++ версия GraphChi. Последующие тесты будут совершены на машине с процессором Intel Core i52430M CPU, 2.40 GHz и 4 GB RAM и 500 GB SATA HDD, поэтому результаты могут отличаться от идеальных. Например, компьютеры с SSD дисками в среднем работают в два раза быстрее, и для некоторых алгоритмов (в том числе Гирвана-Ньюмана) даже превосходят MapReduce [7, 12].

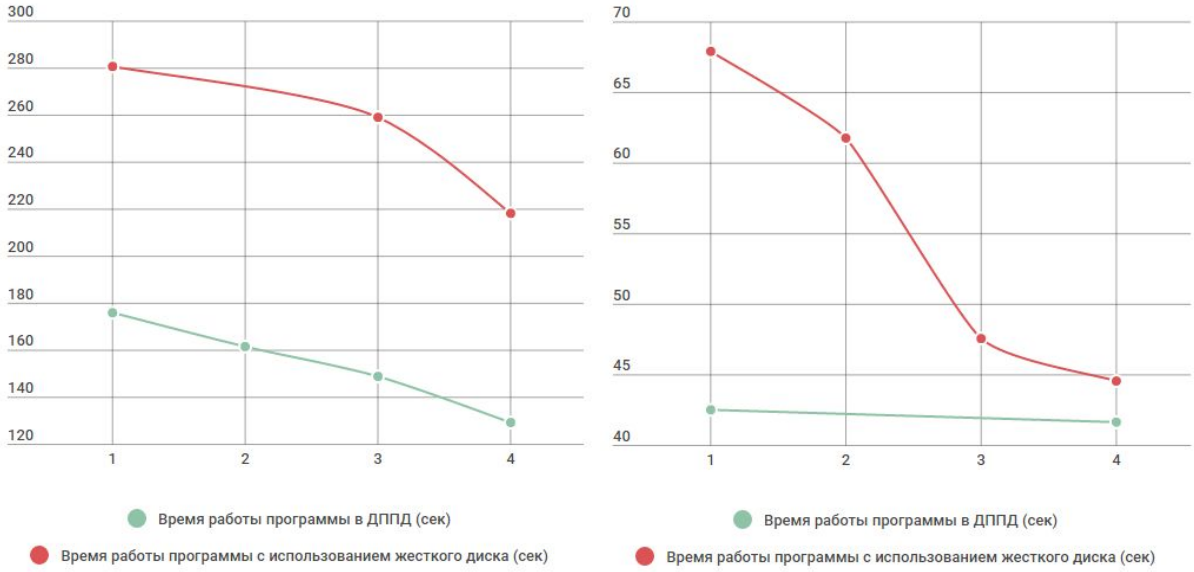


Рисунок 1. Зависимость времени исполнения от количества потоков

На рисунке 1 изображен график зависимости времени исполнения программы в секундах от количества потоков. Первый из них показывает данное соотношение для реализации, использующей алгоритм распространения меток совместно с алгоритмом Гирвана-Ньюмана, второй - только алгоритм распространения меток. Можно заметить, что время работы программы значительно уменьшается при использовании большего числа потоков, показывая эффективность распределенной обработки. В случае использования алгоритма распространения меток запуск программы с четырьмя потоками даже позволяет приблизить эффективность программы, использующей ресурсы HDD к программе, выполняющейся полностью в оперативной памяти.

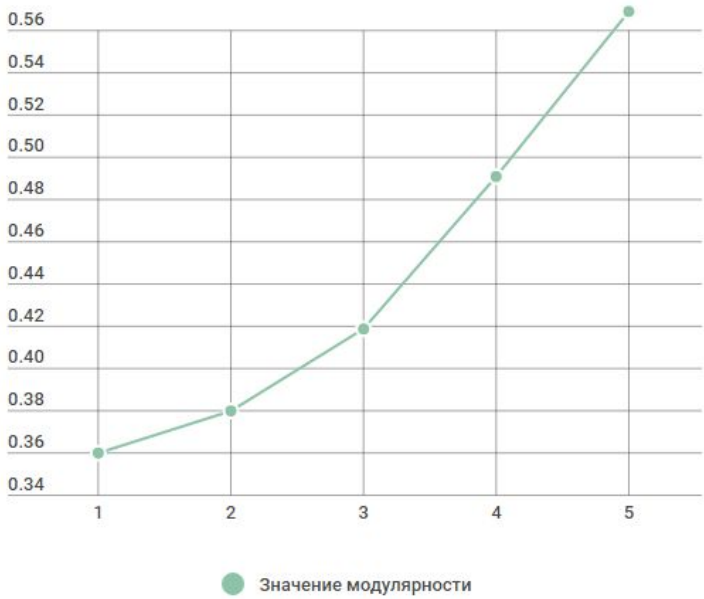


Рисунок 2. Зависимость значения модулярности от количества итераций при совместном использовании LPA и алгоритма Гирвана-Ньюмана

На рисунке 2 показан график зависимости значения модулярности при увеличении количества итераций алгоритма Гирвана-Ньюмана (а значит, и увеличении числа удаленных ребер) с последующим применением алгоритма распространения меток.

1.7.3 Выводы

Из проведенных тестов можно сделать вывод, что алгоритм распространения меток является гораздо более быстродейственным в отдельности, чем в сочетании с алгоритмом Гирвана-Ньюмана, однако его недостатком является то, что качество определения сообществ в данном случае нельзя улучшить. При использовании совместного подхода качество результата может быть улучшено, что выражено в графике на рисунке 2. А также было замечено, что распределенная обработка, даже при запуске программы с использованием ресурсов жесткого диска, позволяет приблизить эффективность программы к эффективности работы в оперативной памяти.

Глава 2. Текстовая классификация и тематическое моделирование

Для практической реализации был выбран язык программирования Python версии 3.5, так как он содержит огромное количество уже готовых, оптимизированных библиотек для решения поставленной задачи. К тому же выбранный язык может предложить не только обучающие алгоритмы, но и множество удобных функций для обработки данных.

2.1. Предварительная обработка текстовой информации

Предварительная обработка поступающей информации является важнейшим этапом как текстовой классификации, так и тематического моделирования (в особенности для русского языка, в котором присутствует огромное количество различных словоформ).

Основными этапами обработки являются:

- Токенизация: разбиение документа на атомарные единицы.
- Нормализация: приведение слов к начальной форме (‘бежал’ - ‘бежать’)
- Удаление стоп-слов, не несущих смысловой информации.
- Стемминг (опционально): нахождения основы слова, что позволяет в дальнейшем рассматривать разные слова с одинаковым смыслом как одно и то же слово (например, ‘бежать’ и ‘убежать’).

1. Токенизация

Текст в электронном виде представляет собой последовательность символов. Для качественной обработки текст должен быть разделен на отдельные лингвистические единицы, такие как слова, числа и др. Этот процесс называется **токенизацией**. Конкретный вид токенов очень сильно зависит от задачи, однако любое разбиение должно подчиняться двум следующим правилам:

1. Токен должен лингвистически иметь смысл
2. Разбиение должно быть методологически применимо [24]

В данной работе на сообщениях, полученных из социальной сети, в качестве первого этапа была произведена токенизация с помощью **регулярных выражений -** стандартной нотации для описания последовательностей символов**.** Регулярные выражения используются для определения символьных строк по *шаблонам* (например, для поиска всех слов в тексте, начинающихся с определенного слога) и часто применяются в веб-поиске, в задачах информационного поиска, а также играют важную роль в обработке естественных языков [24]. Для работы с регулярными выражениями на языке *Python* используется стандартная библиотека *re*. С помощью данной библиотеки из сообщений были удалены гиперссылки и прочий информационный мусор, а затем тексты были разбиты на слова по не буквенным символам.

Далее все слова были приведены к нижнему регистру с помощью встроенной функции *lower*.

2. Нормализация

Для **нормализации,** т.е. приведения слова к начальной форме (например, для существительных - это единственное число, именительный падеж), была использована библиотека *pymorphy2,* разработанная специально

для русского и украинского языков. Библиотека использует лексемы, взятые из таких открытых ресурсов как *OpenCorpora* и *LanguageTool* data, которые специально обработаны для быстрого поиска, а также набор лингвистических правил для обеспечения морфологического анализа и обработки слов, отсутствующих в словаре. *pymorphy2* распространяется под лицензией *MIT*. [26]

3. Удаление стоп-слов

Стоп-словами называют часто встречающиеся в текстах слова (различные предлоги, местоимения, междометия и др.), которые мы хотели бы исключить из рассмотрения. Такие слова содержат мало самостоятельного лексического смысла, и их присутствие в тексте только затрудняет задачу классификации. [25]

В языке *Python* существует две известные библиотеки, содержащие список стоп-слов для русского языка:

- *NLTK* - одна из самых широко используемых платформ для работы с человеческими языками. Библиотека обладает огромным функционалом для предварительной обработки текста со встроенными инструментами для токенизации, стемминга, а также встроенными корпусами для обучения и анализа, и др. [28]
- *stopwords* представляет собой исключительно набор списков стоп-слов для различных языков.

Было установлено, что в списке *NLTK* содержится 151 стоп-слово, а в наборе ***stopwords*** - 421, поэтому для использования в работе была выбрана именно библиотека ***stopwords***.

4. Стемминг

Стемминг — это процесс нахождения основы для исходного слова (основа слова необязательно совпадает с морфологическим корнем).

Большинство алгоритмов стемминга были созданы для английского языка, однако на сегодняшний день существует несколько разработок и для русского языка. Русский язык имеет сложную морфологическую изменяемость слов, которая является источником ошибок при использовании стемминга. В качестве решения данной проблемы можно использовать наряду с классическими алгоритмами стемминга алгоритмы лемматизации, которые приводят слова к начальной форме, что и было сделано ранее (п.2).

Для практической реализации был использован *Snowball*-стеммер из библиотеки *NLTK*. *Snowball* - язык программирования, придуманный М. Портером, широко используемый для стемминга слов на различных языках. В частности, в русском языке, алгоритмы такого типа используют отсечение словообразующих суффиксов, а затем применяют различный набор правил для нахождения основы слова (например, ‘в основе слова на русском языке обязательно присутствует хотя бы одна гласная’). Данная технология проста в применении, быстродейственна, однако зачастую приводит к ошибкам, особенно в русском языке. [29]. В данной работе было использовано два варианта предварительной обработки - со стеммингом и без него.

2.2. Обзор подходов к классификации документов

Как было упомянуто во введении, одной из больших проблем в работе с социальными данными является огромное количество поступающей информации. Поэтому классификация документов, а также выделение среди всех документов именно тех, которые интересуют исследователей - одна из важнейших задач анализа социальных данных.

К решению данной задачи существует несколько основных подходов:

1. **Классификация, основанная на правилах**. В данном подходе пользователь определяет правила, по которым производится разбиение документов (Напр. “Если текст содержит слова ‘определитель’, ’дифференциал’, отнести его к категории ‘высшая математика’”).

- ***Достоинства*:** Данный подход показывает хорошую точность при небольших наборах документов для классификации. Результаты всегда предсказуемы, так как правила определяются явно самим пользователем.
- ***Недостатки*:** Правильное определение правил для больших коллекций документов - трудоёмкий и непростой процесс. При увеличении количества текстов, может увеличиться и необходимое количество правил для их классификации.

Подходы, основанные на машинном обучении

2.1 **Обучение с учителем.** В этом подходе набор правил определяется автоматически по обучающим данным.

- ***Достоинства***: Правила определяются программой автоматически, что может быть полезно для больших коллекций документов.
- ***Недостатки*:**
    - Необходимо заранее вручную распределить документы обучающей выборки по категориям.
    - Правила не доступны пользователю явно и могут оказаться не такими, как предполагается (возможность *переобучения*).

2.2. **Обучение без учителя.** Известны только описания множества объектов (тестовой выборки), и требуется автоматически обнаружить

внутренние взаимосвязи, зависимости, закономерности, существующие между объектами.

- ***Достоинства*:**
    - Нет необходимости предоставлять ни правила, ни обучающее множества для осуществления кластеризации.
    - Помогает обнаружить закономерности, которые пользователь мог упустить.
    - Часто используется для построения начального разбиения на категории с последующим применением других методов.
- ***Недостатки*:**
    - Результатом кластеризации могут стать совсем неожиданные группы, так как принципы объединения задаются полностью программно.
    - Кластеризация требует большого количества процессорных операций.

2.2.1 Байесовский классификатор

В силу того, что одна из составляющих частей работы - это обработка большого числа социальных данных - был выбран алгоритм машинного обучения, так как обработка больших данных с помощью подхода, основанного на правилах, представляет собой нетривиальную и\или вычислительно-ёмкую задачу. Методы, основанные на обучении без учителя, были так же отклонены в силу их неточной обработки сырых первоначальных данных. В качестве основного метода обработки был использован алгоритм машинного обучения с учителем - Мультиномиальный наивный байесовский классификатор (МНБК), так как данный метод хорошо

показывает хорошую точность в работе с большим количеством классов (> 2, с точностью до 95%), а также обладает простотой математической модели и высокой скоростью. [27] Принцип его работы будет описан далее.

Пусть количество категорий - C, размер словаря - N, тогда МНБК определяет документ t_i в категорию, имеющую наибольшее значение вероятности $Pr(c|t_i)$, которая согласно формуле Байеса имеет следующий вид:

$$Pr(c|t_i) = \frac{Pr(c)Pr(t_i|c)}{Pr(t_i)},\ c \in C\text{, где}$$

1. $Pr(c)$ вычисляется как отношение количества документов, принадлежащих классу *c* к общему количеству документов.

2. $Pr(t_i|c)$ - вероятность встретить документ, похожий на t_i в классе *c*, которая вычисляется следующим образом:

$$(\sum_n f_{ni})!\Pi\frac{Pr(w_n|c)^{f_{ni}}}{f_{ni}!}\text{, где} \qquad (2.3.1)$$

f_{ni} - количество раз, которое слово n встретилось в документе t_i

$Pr(w_n|c)$ - условная вероятность встретить слово w_n в документах класса *c,* оцениваемая по формуле:

$$Pr(w_n|c) = \frac{1 + F_{nc}}{N + \sum\limits_{x=1}^{N} F_{xc}}\text{, где}$$

F_{xc} - количество раз, которое слово x встречалось во всех документах класса C (к числителю также применено аддитивное сглаживание для решения проблемы неизвестных слов)

Можно заметить, что в уравнении (2.3.1) трудно вычислимые множители $(\sum_n f_{ni})!$ и $\Pi f_{ni}!$ не зависят от класса C, поэтому обычно их не учитывают, на практике вычисляя вероятность по формуле:

$$Pr(t_i|c) = \alpha\Pi Pr(w_n|c)^{f_{ni}}$$

3. $Pr(t_i)$ вычисляется по следующей формуле:

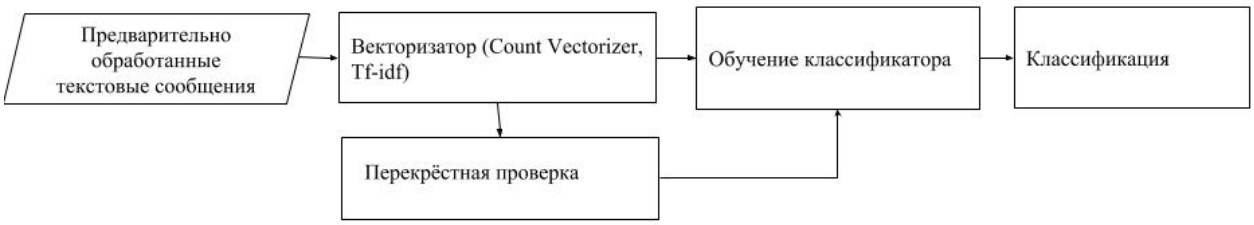
$$Pr(t_i) = \sum_{k=1}^{|C|} Pr(k)Pr(t_i|k)$$

2.2.2. Практическая реализация разделения сообщений на категории

Первоначальной задачей было составление обучающей выборки для наивного байесовского классификатора. Был получен набор, состоящий из 1200 записей из социальной сети ВКонтакте со стен студентов факультета ПМ-ПУ университета СПбГУ и затем вручную разбит на следующие **категории**: искусство, бизнес, музыка, университет, технологии, личная жизнь, здоровье, наука, политика, спорт и рекламные записи.

Далее на основе обучающей выборки алгоритм классифицирует поданные в него записи и распределяет их по 11 вышеупомянутым категориям. В программе это реализовано с помощью библиотеки, разработанной и оптимизированной для решения задач машинного обучения, *scikit-learn*. Библиотека хорошо документирована, имеет простой и понятный API, а также содержит множество инструментов для классификации, кластеризации, моделирования и предварительной обработки данных. [30] *scikit-learn* распространяется под лицензией *BSD*, т.е. является *open-source* программным обеспечением.

Процесс классификации можно разделить на несколько **этапов**:



Первоначально все документы были обработаны по схеме, описанной в параграфе 2.1.

1. Векторизация

На данном этапе каждый документ должен быть представлен в векторном виде, с которым может работать классификатор. Для осуществления данного представления в работе рассматривалось два подхода:

1. мешок слов (*bag-of-words*)
2. разбиение на N-граммы

Модель bag-of-words.

В модели *bag-of-words* документ представляется в виде набора пар “слово” - “число”, где число - количество раз, которое слово встречалось в тексте (его *признак*). В данной модели предполагается, что порядок слов в предложении не важен - важна только частота появлений конкретного слова. Набор всех таких пар, называемый *словарём*, создается из всех документов, присутствующих в корпусе. Каждый документ представляется в виде вектора d с целочисленными элементами, где d_i - частота появления слова i в документе d. Объединяя вектора, мы получаем таблицу, содержащую частоты всех слов, присутствующих в текстах, называемую *терм-документной матрицей*.

В *scikit-learn* для осуществления данного представления документов существует класс *CountVectorizer,* с помощью функций которого, по корпусу документов строится терм-документная матрица.

Для увеличения точности классификации была использована **Tf-Idf** трансформация. *Tf-Idf* - статистическая мера, используемая для оценки важности слова в документе, являющемся частью корпуса.

При применении данной трансформации в качестве веса слова используется не количество раз, которое оно появлялось в тексте, а значение, вычисленное по формуле:

$$\text{TFIDF(word)} = log(f + 1)log(\frac{D}{df}) \text{ , где}$$

f - частота употребления слова

df - количество документов, содержащих рассматриваемое слово

D - общее количество документов [27].

Таким образом, вес некоторого слова в рамках документа пропорционален количеству его появлений в данном документе, и обратно пропорционален частоте его употребления в других документах корпуса.
Данную функцию в *scikit-learn* выполняют методы класса *TFIDFVectorizer*.

N-граммы

Последовательность из N идущих друг за другом слов называют **N-граммой**. В отличие от *bag-of-words* в данной модели предполагается, что слова в тексте не независимы, а зависят от предыдущих N слов [24].
Основной недостаток применения N-грамм, это быстро растущий объем памяти необходимый для их обработки, обеспечивающий существенное снижение скорости работы программы. [29] В данной работе рассматривалось разбиение текста на биграммы и триграммы, каждая из которых считалась одним токеном. Это было осуществлено с помощью отдельного конструктора для класса *CountVectorizer*, в котором задавался необходимый размер N-грамм.

2. Перекрестная проверка

Перекрестной проверкой (*cross-validation*) называется методика, смысл которой заключается в разделении обучающей выборки на две части. Первая передается алгоритму вместе с правильными ответами и используется для обучения. После этого алгоритм прогнозирует категории для экземпляров из второй части. Полученные ответы сравниваются с правильными, и вычисляется суммарная оценка качества алгоритма. Однако при таком простом разделении результат оценки может сильно зависеть от того, какие данные попали в обучающую, а какие в тестируемую выборку. В *k-проходной перекрестной валидации* исходная выборка разделяется случайным образом на *k* равных частей. Одна из этих частей используется для тестирования, *k*-1 - для обучения классификатора. Данный процесс повторяется *k* раз (проходов), из которых далее берется средняя оценка качества классификации.

Для проведения *k*-проходной перекрестной валидации был использован класс *Kfold* библиотеки *scikit-learn.*

3. Обучение классификатора и классификация

Для классификации сообщений был использован класс *MultinominalNB* библиотеки *scikit-learn*.

2.5. Тематическое моделирование

В предыдущем параграфе было рассмотрено разбиение сообщений по тематическим, заданным заранее, категориям с помощью классификатора. В данном параграфе будет рассмотрена задача тематического моделирования.

Тематическая модель коллекции текстовых документов определяет, к каким темам относится каждый документ и какие слова образуют каждую тему.

Вероятностная тематическая модель описывает каждую тему дискретным распределением на множестве терминов, каждый документ — дискретным распределением на множестве тем. Предполагается, что коллекция документов — это последовательность терминов, выбранных случайно и независимо из смеси таких распределений, и ставится задача восстановления компонент смеси по выборке. Каждый документ или термин может относиться одновременно ко многим темам с различными вероятностями. [29]

2.5.1. Латентное размещение Дирихле

Одна из самых распространенных и широко используемых вероятностных тематических моделей - Латентное размещение Дирихле (*LDA*). [29]

Основная идея *LDA* заключается в том, что документы представляются смесью распределений латентных тем, где каждая тема определяется вероятностным распределением на множестве слов. [32] Другими словами, предполагается, что каждый текст содержит в себе несколько тем, а также, что каждое слово связано с какой-либо темой документа.

Графическое представление модели *LDA* показано на рисунке 3.

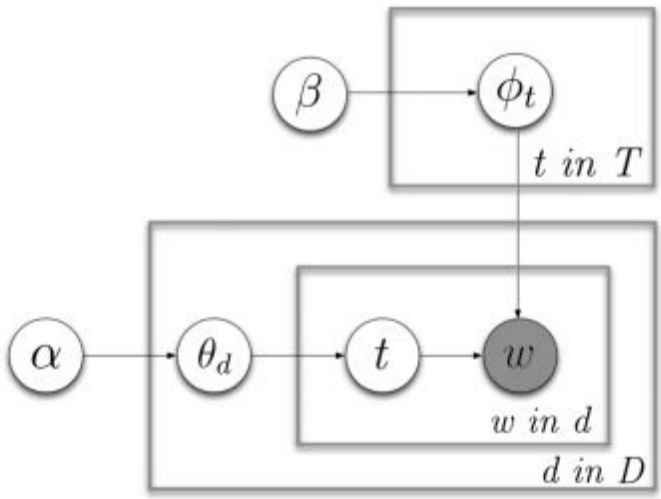


Рисунок 3

Данный метод можно описать в виде трех шагов:

1. Для документа d выбирается случайный вектор θ_d из распределения Дирихле с параметром α.
2. Выбирается тема z_{di} из мультиномиального распределения с параметром θ_d .
3. Согласно выбранной теме z_{di} выбирается слово w_{di} из распределения Φz_{di} , которое является распределением Дирихле с параметром β

Таким образом, порождающая модель слова *w* из документа *d* представляется в виде:

$$p(w|d,\theta,\Phi) = \sum_{t=1}^{T} p(w|z,\Phi_z)p(z|d,\theta_d)$$

Для оценки параметров LDA используются простой вариационный EM-алгоритм и семплирование Гиббса [32].

Основным недостатком модели, основанной на распределении Дирихле, несмотря на её частое использование, является отсутствие убедительных лингвистических обоснований [29].

2.5.2. Критерий качества модели

Тематическое моделирование представляет из себя задачу “мягкой” кластеризации, поэтому оценить качество построенной модели программно не всегда просто. Несмотря на это, для анализа качества тематических моделей используется несколько критериев. Среди них наиболее важным является **степень неопределенности** (perplexity) [32]. Этот критерий используется также для оценки обобщающей способности модели на тестовых данных и нахождения оптимального числа тем в корпусе. Одним из достоинств данного критерия является то, что его вычисление не требует предварительной категоризации документов.

Для тестовой выборки из N документов степень неопределенности определяется равенством:

$$Perplexity = e^{-\frac{\sum_{d=1}^{M} ln(p(w_d))}{\sum_{d=1}^{M} N_d}}$$

Чем меньше степень неопределенности в тестовой коллекции документов, тем лучше обобщающая способность построенной модели.

2.6. Практическая реализация тематического моделирования

Для практической реализации в рамках данного инструментария была выбрана библиотека *gensim*, написанная для языка программирования *Python* [31]. Данная библиотека широко используется для решения задач тематического моделирования, содержит огромное количество готовых моделей, представляет хорошо масштабируемые решения (есть параллельные

версии некоторых алгоритмов), а также имеет хорошую документацию, поэтому в качестве решения была выбрана именно она.

1. **Предварительная обработка**

Первоначально все тексты были обработаны по схеме, описанной в п.2.1 за исключением стемминга, так как результаты, полученные с использованием стемминга, были намного менее понятны для человеческого восприятия.

2. **Векторизация**

Затем каждый документ был представлен в векторном виде с использованием модели *bag-of-words* (п.2.2.2). Для построения терм-документной матрицы на этот раз была использована функция *doc2bow* класса *Dictionary* библиотеки *gensim.*

3. **Построение модели**

Для составления модели был использован класс *LdaModel* библиотеки *gensim.*

Для определения оптимального количества тем в каждой категории, а также для оценки качества модели была использована степень неопределенности (*perplexity*). Количество тем изменялось с некоторым шагом, при этом было произведено построение модели, для которой в последствии вычислялась степень неопределенности. Затем определялось количество тем, при котором значение неопределенности было минимальным, что является стандартным подходом для оценки параметров модели LDA.

Для расчета степени неопределенности использовалось готовое решение, присутствующее в библиотеке *gensim* - *log_perplexity,* находящаяся в классе *LdaModel*. Конечное значение степени неопределенности высчитывалось по формуле: $Perplexity = 2^{-log\ perplexity}$ для приведения к более удобному виду.

Глава 3. Тестирование работы программы

В качестве примера для иллюстрации работы данного инструментария была собрана небольшая социальная сеть подписчиков группы “ПМ-ПУ СМИ”. На данной социальной сети были выделены сообщества, а также произведена текстовая классификация и тематическое моделирование сообщений, оставляемых подписчиками у себя на стене. В результате работы программы была получена информация, из которой можно сделать вывод об интересах тех или иных групп пользователей.

3.1. Сбор информации из социальной сети ВКонтакте

Для получения данных пользователей социальной сети ВКонтакте была использована библиотека *requests*. Библиотека *requests* позволяет отправлять HTTP-запросы HEAD, GET, POST, PUT, PATCH и DELETE. [23] С помощью данной библиотеки совершались GET-запросы к API ВКонтакте следующего вида: ‘https://api.vk.com/method/*METHOD*’, где вместо *METHOD* указывался тип информации, которую необходимо получить (использованные методы описаны в таблице 1).

Сервер возвращает ответ в формате json, который обрабатывается программой и сохраняется в SQL базу данных, работа с которой в *Python* обеспечивается библиотекой *sqlite3*.

При достижении программой ограничения API ВКонтакте (более 3 запросов в секунду), она делает паузу в 1 с, а затем продолжает работу.
Все используемые в работе методы открытые, т.е. не требуют *access_token*.
Процесс построения социальной сети описан в таблице 1.

Таблица 1. Описание использованных методов API и их параметров

Название метода	**Описание параметров**	**Как использовалось**
1. groups.getMembers Возвращает список идентификаторов пользователей	• *group_id* - идентификатор или короткое имя сообщества.	Для получения идентификаторов участников сообщества “ПМ-ПУ СМИ”
2.friends.get Возвращает список идентификаторов друзей пользователя или расширенную информацию о друзьях пользователя (при использовании параметра *fields*)	• *user_id* - идентификатор пользователя, для которого необходимо получить список друзей • *fields* - список дополнительных полей, которые необходимо вернуть (в нашем случае *sex*, *city, home_town*)	Для получения списка друзей каждого из полученных в п.1 пользователей, также входящих в группу “ПМ-ПУ СМИ”. С помощью данного метода был создан социальный граф, который был сохранен в текстовом файле в виде списка рёбер, так как этот формат подходит для использования в GraphChi, а также является самым компактным по занимаемой памяти. Вся информация о пользователях была сохранена в базу данных
3.database.getCitiesById Возвращает информацию о городах по их идентификаторам	• *city_ids* - идентификаторы городов	Для получения названия города (поле ‘name’) по его ID, полученном на предыдущем шаге.
4.wall.get Возвращает список записей со стены пользователя или сообщества.	• *owner_id* - идентификатор пользователя или сообщества, со стены которого необходимо получить записи • *count* - количество записей, которое необходимо получить • *filter=owner* - определяет, какие типы записей на стене необходимо получить (в нашем случае только от владельца) • *offset* - смещение, необходимое для выборки определенного подмножества записей	Для получения списка *count* сообщений со стены каждого пользователя, полученного в п.1. *count* был установлен равным 50. Одной из трудностей при сохранении сообщений оказалось то, что пользователи очень часто сохраняют у себя на стене изображения без какой-либо текстовой информации. Решением послужило использование цикла, игнорирующего записи без текста, работающего пока не наберутся требуемые 50 сообщений, с использованием параметра *offset*. В разработанной программе ему присваивается число удачно прочитанных записей со стены (содержащих текстовую информацию)

Таким образом, сперва был получен список подписчиков сообщества “ПМ-ПУ СМИ” во ВКонтакте. Затем для каждого подписчика был получен список его друзей - и среди них были выбраны те, которые состоят в сообществе “ПМ-ПУ СМИ”. В итоге была получена социальная сеть в виде графа из ~1300 участников и более 50 000 связей между ними (1293 вершин и 50016 рёбер). Затем, для каждого участника сети были получены записи с его стены. В итоге было получено около 20 000 текстовых сообщений пользователей.

3.2. Обнаружение сообществ в полученной социальной сети

Полученная социальная сеть в виде графа была обработана с помощью созданного алгоритма в GraphChi. В результате работы программы граф был разбит на 11 сообществ и раскрашен цветами для наглядности. Информация о самых крупных сообществах представлена в таблице 2, а также результат работы программы на GraphChi в формате *gdf* визуализирован с помощью программы Gephi (рисунок 1).

Таблица 2. Количество пользователей в полученных сообществах

ID сообщества	**Число участников**	**ID наиболее влиятельного члена сообщества**
1	411	217
2	189	373
3	144	880
5	339	19
9	175	711
Остальные сообщества	35	504

Как показала проверка, наиболее влиятельные члены сообществ - это пользователи с большим количеством друзей во ВКонтакте, принимающие активное участие в жизни факультета.

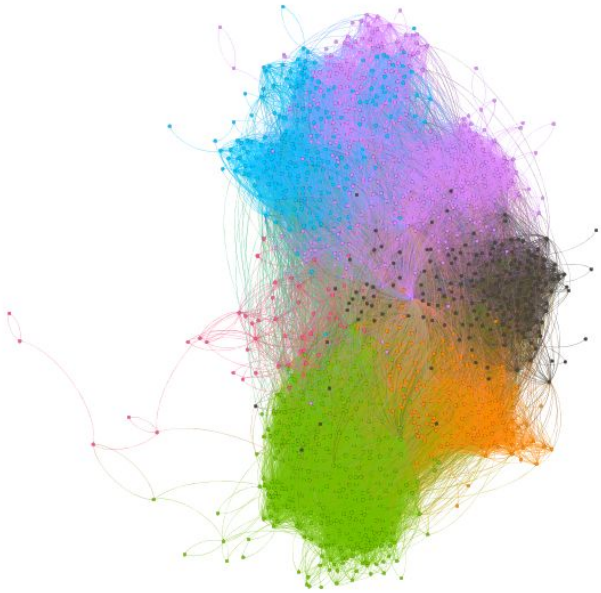


Рисунок 1. Социальная сеть, разбитая на сообщества

В таблице 3 описана конфигурация использованного алгоритма.

Таблица 3. Конфигурация алгоритма обнаружения сообществ

Режим работы	**Число потоков**	**Время работы**	**Значение модулярности**
in-memory	4	1 мин. 38 сек.	0,523
disk-based	4	9 мин. 56 сек.	0,522

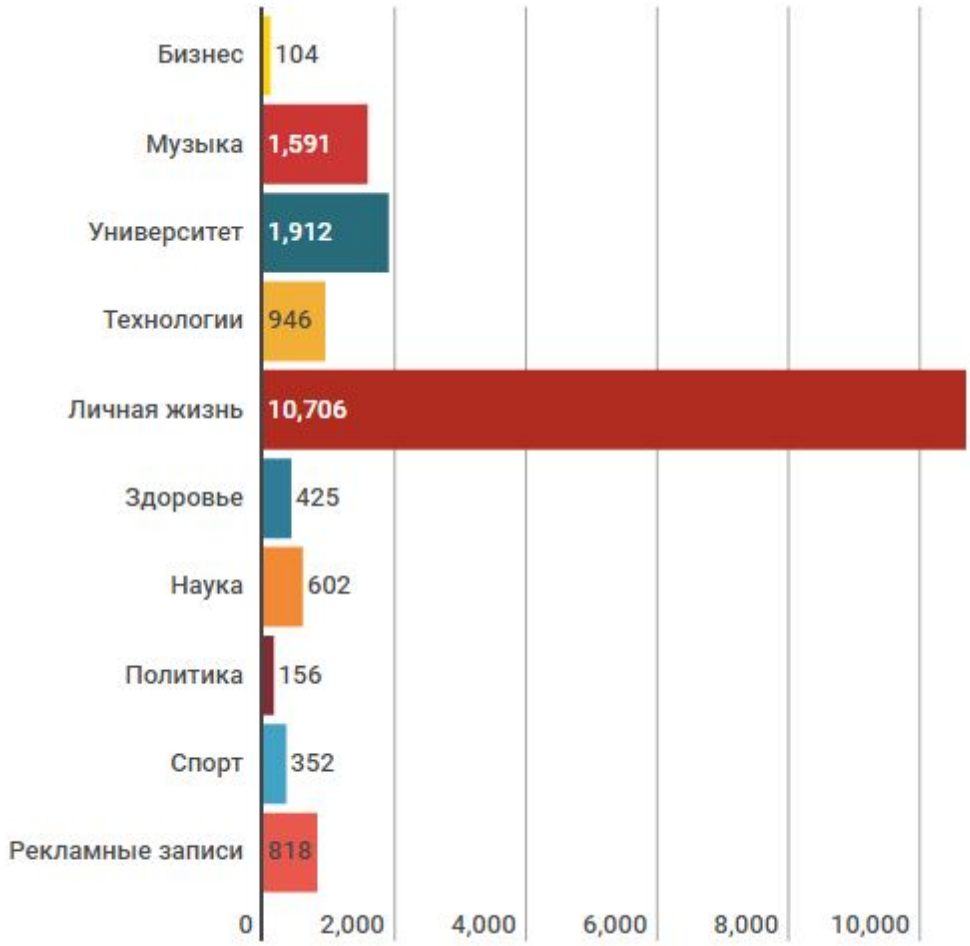
3.3. Текстовая классификация и тематическое моделирование сообщений пользователей полученной сети

Полученные 20 000 текстовых сообщений членов сообщества “ПМ-ПУ СМИ” (тестовая выборка) была классифицирована обученным заранее на

выборке из 1200 сообщений мультиномиальным наивным байесовским классификатором (подробно описан в главе 2).

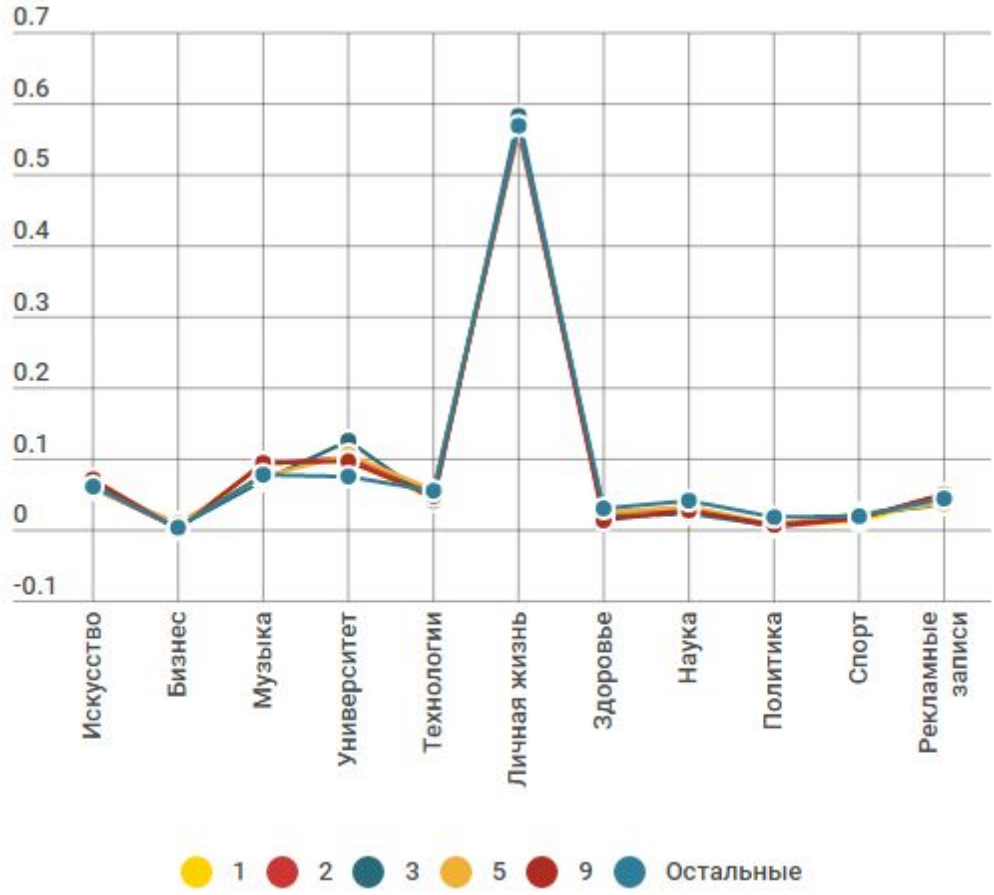
3.3.1. Результаты классификации

В результате классификация первоначальная выборка была разбита на 11 **категорий**: искусство, бизнес, музыка, университет, технологии, личная жизнь, здоровье, наука, политика, спорт и рекламные записи. Распределение всех имеющихся сообщений по категориям можно увидеть на столбцовой диаграмме:



Распределение тематик сообщений по сообществам можно увидеть на графике 1. На оси абсцисс отмечены тематические категории, на оси ординат - отношение сообщений, принадлежащих определенной категории, к общему

числу сообщений пользователей данного сообщества. Графики для различных сообществ отмечены различными цветами.



Можно заметить, что распределение сообщений по категориями для всех сообществ примерно одинаково, что скорее всего объясняется тем, что все рассмотренные пользователи являются студентами одного факультета, приблизительно одного возраста, а также проживающие в одном городе.

В том числе был сделан вывод о том, что участники одного сообщества не обязательно должны иметь общие интересы, хотя определенная зависимость в обратном направлении прослеживается. Например, было установлено, что в сообществе 9 - максимальная среди остальных сообществ доля сообщений в категориях: искусство, музыка, рекламные записи. Возможно, это связано с тем, что во ВКонтакте часто производятся розыгрыши билетов на

музыкальные фестивали и концерт. Также в 9 сообществе минимальная среди других доля сообщений категорий бизнес и здоровье.

В сообществе 5 максимальный по отношению к другим сообществам процент сообщений категорий бизнес и технологии, минимальная же доля в категории искусство. Следовательно, можно сделать вывод о том, что члены пятого сообщества - любители информационных технологий и мира IT в целом.

В 3 сообществе максимальный по отношению к другим сообществам процент достигается в категориях личная жизнь и университет. Вполне возможно, что члены третьего сообщества - это люди, принимающие участие в жизни университета.

3.3.1. Оценка качества классификации

1. *k-проходная перекрестная проверка*

Первоначальная оценка классификации была получена при осуществлении перекрестной валидации. После проведения k проходов была получена матрица ошибок (confusion matrix). Среднее качество классификации составило ~87 %. Наиболее часто классификатор путал категории “Спорт” и “Здоровье”, а также “Наука” и “Технологии”. Вполне вероятно, это произошло потому, что в этих категориях часто встречаются общие между ними слова.

2. Тестирование качества работы классификатора

Для тестирования были случано выбраны 500 текстовых сообщений. Выборочная проверка показала, что точность классификации на тестовых данных составляет ~75%.

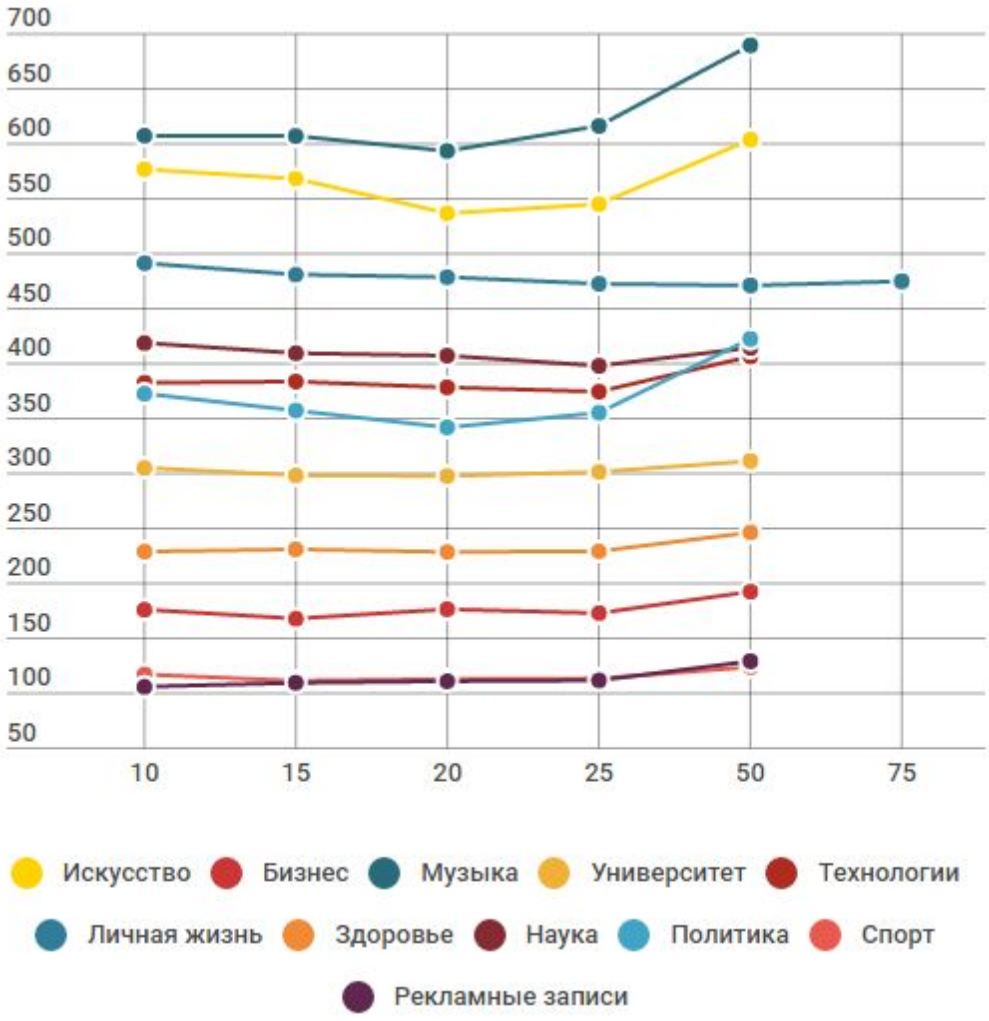
3.3.2. Результаты тематического моделирования

Для определения параметров модели, а именно количества тем, первоначально был проведен анализ степени неопределенности (описан в параграфе 2.5.2.). Результаты можно увидеть на графике и в таблице 4. Категории в таблице пронумерованы согласно порядку в легенде графика.

Таблица 4. Оптимальное количество тем в каждой категории

# кат	1	2	3	4	5	6	7	8	9	10	11
тем	20	15	20	20	25	50	20	25	20	15	10

График степени неопределенности для документов каждой категории



Далее были составлены тематические модели с установленными параметрами для каждой из вышеупомянутых категорий. Результат в виде списка слов для каждой из тем представлен в приложении 1.

Выводы

В ходе исследования был получен инструментарий, позволяющий упростить анализ сложно связанных социальных данных путём разделения их на более простые составляющие и выполнения автоматизированной обработки составляющих как отдельных независимых частей системы. Это было достигнуто путём реализации алгоритмов разделения социальных графов на сообщества, в частности алгоритма Гирвана-Ньюмана и алгоритма распространения меток. Было установлено, что алгоритм распространения меток является значительно более быстродейственным, однако, во многих случаях его совместное применение с алгоритмом Гирвана-Ньюмана позволяет значительно улучшить полученный результат. Для определения наиболее влиятельного члена каждого сообщества был использован PageRank каждого узла в социальном графе. Реализация данных алгоритмов значительно упростила понимание структуры сети в целом и позволила определить наиболее влиятельных пользователей каждого сообщества.

Также для анализа характерных признаков пользователей с использованием сообщений, оставляемых ими в социальной сети, был использован готовый алгоритм текстовой классификации (мультиномиальный байесовский классификатор), предоставляемый библиотекой для машинного обучения на языке *Python*. Для построения

тематической модели каждой из полученных категорий было использовано латентное размещение Дирихле. Полученные модели позволили, в целом, оценить тематики, наиболее волнующие пользователей по каждой из составленных категорий.

Для тестирования работы приложения было выбрано одно из сообществ в соцсети ВКонтакте. Работа инструментария показала, что он действительно обеспечивает упрощение анализа сложно связанных данных, так как в результате компьютерной обработки 20 000 текстовых сообщений, была быстро получена необходимая информация. Можно с уверенностью сказать, что все задачи, поставленные в этой работе, были выполнены.

Заключение

В ходе выполнения ВКР был создан инструментарий, позволяющий собирать и обрабатывать сложно связанные социальные данные.

В ходе работы были решены задачи:

1. Анализ основных подходов к обнаружению сообществ в социальных графах.
2. Реализация алгоритма обнаружения сообществ среди участников социальной сети, позволяющего обрабатывать даже большие объемы данных.
3. Выбор и реализация алгоритмов текстовой классификации и тематического моделирования.

Полученные результаты могут быть использованы для развития систем рекомендаций, изучения определённых групп или слоёв населения и решения схожих проблем. В дальнейшем данный инструментарий может быть улучшен и дополнен путём внедрения функций идентификации пользователя (обнаружение аккаунтов, принадлежащих одному человеку), измерения информационного влияния и др.

Приложение 1

Для удобства восприятия из каждой темы были выбраны слова с максимальным значением вероятности, а также приведены только некоторые из тем.

Таблица 5. Результат тематического моделирования

Категория	**Тематические слова, полученные с помощью LDA**
Искусство	● эрмитаж, шедевр, выставка, коллекция, музей ● фильм, миллион, сумерки, майер, фанат, фэнтези ● фильм, оскар, остров, актёр, леонардо, режиссёр ● серия, дарт, вейдёр, люк, душа, рука ● фильм, гарри, битва, поттер, сражение ● история, сезон, серия, джон, песнь ● шедевр, книга, идея, собрание, классика
Бизнес	● проект, деньги, портфель, сумма, заём, система ● работа, рабочий, рубль, зарплата, знание, лекция ● бизнес, книга, роберт, бодо, шефер, стиль, клиент ● работа, петергоф, парк, площадь, кампания, анализ ● курс, начало, получение, основа, новичок, качество, эксперт
Музыка	● группа, концерт, альбом, голос, вокал, тур ● билет, рок, концерт, февраль, рубль, репост ● гитара, альбом, группа, дуэт, часть ● концерт, группа, альбом, музыка, песня, видео ● голос, выход, николь, генри, интервью, джаз ● рок, июнь, выпуск, инди, альбом, настоящее, атмосфера ● группа, москва, джаз, мастер, звезда, фестиваль
Университет	● команда, спбгу, турнир, игра, матч, победа ● выезд, вуз, список, анкета, друг, новость, база

	• студент, вуз, программа, список, диплом, сайт • курс, язык, клуб, лекция, программа, работа • вопрос, спбгу, сессия, пунк, экзамен, неделя • неделя, фестиваль, факультет, подарок, мистер, пу, пм
Технологии	• ноутбук, часть, рубль, кнопка, ремонт, экран • ракета, проект, танк, армия, модель, часть, война, генерал, огонь • телефон, поддержка, система, камера, платформа, проблема • сайт, ссылка, сеть, сервис, канал, ресурс • кинопоиск, дизайн, яндекс, сайт
Личная жизнь	• любовь, девушка, сердце, глаз, рука, счастье • любовь, дом, слово, счастье, мир, женщина, девушка • свобода, гора, надежда, поход, дружба, станция, ожидание, прогулка • ребёнок, мама, отец, папа, сын, мальчик, родитель, малыш, игрушка, взрослый • город, улица, двор, звук, район, крыша
Здоровье	• мышца, углевод, организм, жир, тело, сила, цель, фаза, вес, работа • кожа, волос, масло, маска, лицо, вода • роды, ребёнок, кровь, мозг, плод, матерь, состояние, женщина, плацента, риск • минута, вода, творог, огонь, смесь, желатина • мука, масло, тесто, крем, сахар, яйцо, торт, литр, духовка • кофе, сахар, зуб, ребёнок, вещество, паста • рубль, средство,волос, тысяча, крем • мозг, курение, никотин, память, эмоция, проблема, сигарета, система, чувство, нейрон
Наука	• реактор, мост, шляпа, клетка, секунда, мышление • страна, город, россия, станция, химикат • теория, работа, мир, вселенная, момент, вопрос • наука, закон, планета, звезда, вселенная, космос, взрыв, теория • система, закон, идея, вода, случай, книга

Политика	● путин, крепость, музей, месяц, четверг, слава, район, анаклия, петиция ● россия, страна, власть, народ, статья, депутат ● россия, война, германия, путин, владимир, войско ● россия, буква, регион, иероглиф, чечня, население ● доллар, украина, доход, госсовет, новость, событие ● джеймс, президент, евро, район, налог, крым ● войско, гвардия, вторжение, случай, оккупация, охрана ● ракета, корабль, медведь, боевик, высота ● европа, вылет, москва, мир, полёт, рейс, африка ● курс, группа, украина, дело, россия ● крым, арест, аксенов, утро, рубль, евро ● владимир, су, инцидент, путин, память, турция ● доллар, брикс, новое, крах, майдан, евро
Спорт	● теннис, спорт, игра, сезон, марафон, июнь ● гантель, неделя, штанга, нога, масса, ватрушка ● реал, тренер, финал, стадион, счёт, минута ● метр, круг, гонка, машина, риккардо, трасса ● штанга, пресс, подход, вес, победа, метод ● занятие, зенит, клуб, танец, суббота, группа ● команда, матч, сезон, игра, игрок, футбол ● колено, мышца, спина, нога, спина
Рекламные записи	● место, репост, конкурс, эфир, футболка, подарок ● билет, приз, клуб, репост, розыгрыш , номер ● розыгрыш, группа, приз, репост, удача, билет ● скидка, февраль, билет, рубль, конкурс, заказ ● конкурс, январь, сообщение, репост, концерт, группа ● репост, условие, группа, друг, запись, акция

Список литературы:

1. Social Network Analysis. Theory and Applications. https://www.politaktiv.org/documents/10157/29141/SocNet_TheoryApp.pdf
2. The top 500 sites on the web. http://www.alexa.com/topsites/global
3. Social Network Analysis Group of Stanford University. http://sna.stanford.edu/
4. Stanford Network Analysis Project. http://snap.stanford.edu/
5. А. Коршунов, И. Белобородов, Н. Бузун, В. Аванесов, Р. Пастухов, К.Чихрадзе, И. Козлов, А. Гомзин, И. Андрианов, А. Сысоев, С. Ипатов, И. Филоненко, К. Чуприна, Д. Турдаков, С.Кузнецов. Анализ социальных сетей: методы и приложения
6. Аудитория Вконтакте. https://new.vk.com/page-47200925_44240810
7. A. Kyrola, G. Blelloch, C. Guestrin. GraphChi: Large-Scale Graph Computation on Just a PC. - 2012
8. К. В. Воронцов. Лекции по алгоритмам кластеризации и многомерного шкалирования. - 2007
9. M. E. J. Newman. Modularity and community structure in networks. - 2006
10. K.Kuzmin, M.Chen. Community Detection via Maximization of Modularity and Its Variants. - 2014
11. M. Girvan, M. E. J. Newman. Community structure in social and biological networks. - 2002
12. S.Moon, et al., Parallel community detection on large graphs with MapReduce and GraphChi, Data Knowl.Eng. -2015
13. U. Brandes. On Variants of Shortest-Path Betweenness Centrality and their Generic Computation. - 2008
14. Статья о промежуточности рёбер на AlgoWiki http://algowiki-project.org/ru/%D0%92%D1%8B%D1%87%D0%B8%D1%

81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_betweenness_centrality

15. L. Page, S. Brin. The Anatomy of a Large-Scale Hypertextual Web Search Engine — 1998
16. D.F. Gleich. PageRank beyond the Web. - 2014
17. A Arasu. PageRank Computation and the Structure of the Web: Experiments and Algorithms
18. K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. -2007
19. A.D. Sarma, A.R.Molla, G.Pandurangan, E. Upfal. Fast Distributed PageRank Computation. Theoretical Computer Science Volume 561, Part B, стр. 113–121. - 2015
20. J. Ugander. Balanced Label Propagation for Partitioning Massive Graphs. -2013
21. U.N. Raghavan. Near linear time algorithm to detect community structures in large-scale networks. - 2007
22. Y. Kim, S. Son, H. Jeong. LinkRank: Finding communities in directed networks. - 2009
23. Веб-сайт библиотеки *requests* http://docs.python-requests.org/en/latest/index.html
24. Daniel Jurafsky and James H. Martin. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. - 2009
25. Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. - 2009.
26. Mikhail Korobov. Morphological Analyzer and Generator for Russian and Ukrainian Languages. - 2015

27. AM Kibriya. Multinomial Naive Bayes for Text Categorization Revisited. -2004
28. Веб-сайт библиотеки *NLTK*. http://www.nltk.org/
29. К. В. Воронцов. Вероятностное тематическое моделирование. - 2013
30. Веб-сайт библиотеки *scikit-learn*. http://scikit-learn.org/stable/
31. Веб-сайт библиотеки *gensim.* https://radimrehurek.com/gensim/
32. Ali Daud, Juanzi Li, Lizhu Zhou, Faqir Muhammad. Knowledge discovery through directed probabilistic topic models: a survey. In Proceedings of Frontiers of Computer Science in China. 2010, 280-301. — перевод на русский К. В. Воронцов, А. В. Темлянцев и др. Обзор по вероятностным тематическим моделям